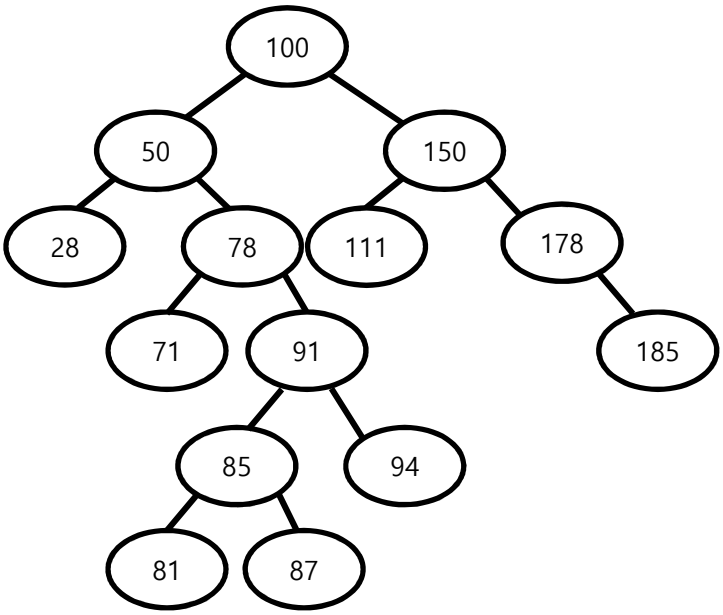
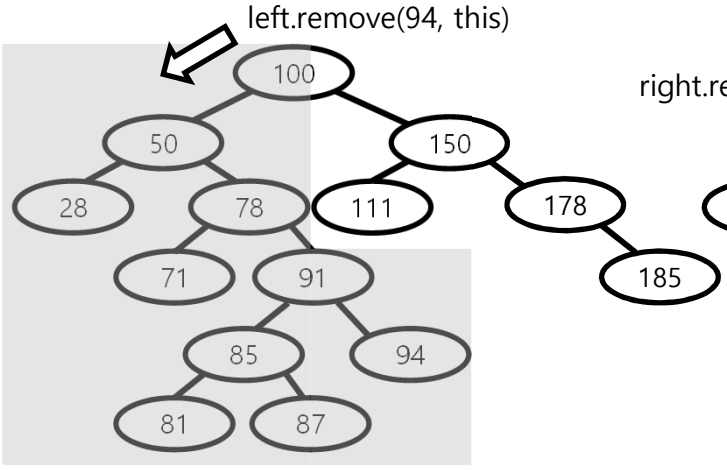
<(첨가1)이진트리 삭제 알고리즘, 어떻게 돌아가는 걸까>

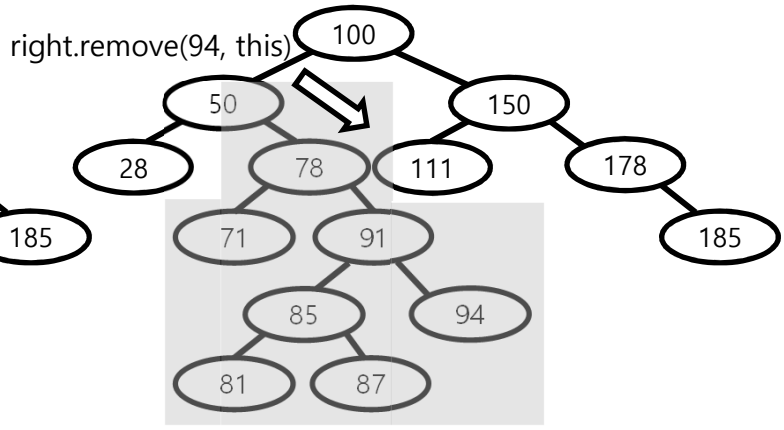



```
public void remove(int value, Node parent){
  if(value<this.value){ // A
    if(left!=null) left.remove(value, this);
  } else if(value>this.value){ // B
    if(right!=null) right.remove(value, this);
  } else{
  if(left!=null && right!=null){ // C
     int temp=right.getLeftMostValue();
     this.value=temp;
     right.remove(temp, this);
  } else { // D
     Node child = (left!=null) ? left:right;
       if(parent.left==this) parent.left=child;
       else parent.right=child;
    }
  }
}
```

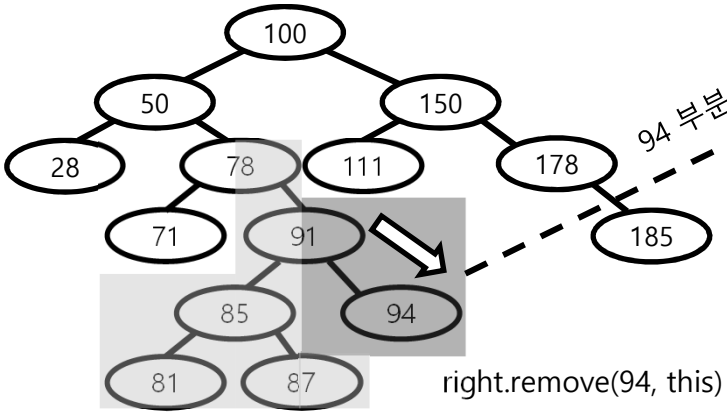
삭제가 되는 과정을 보여주기 위해서 이 트리에 대하여 유지를 하면서 보여주겠다.
1> root.remove(94, root)를 실행하면..



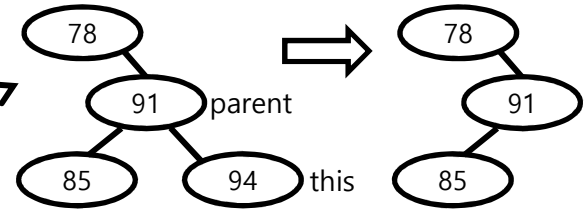
우선 94는 100보다 작기 때문에 A 문장에 의해 실행을 하면서 왼쪽 서브트리에 대해 재귀 호출을 한다.



그 다음 94는 50보다 한참 크기 때문에 B 문장에 의해 오른쪽 서브트리에 대해서 재귀 호출을 하게 된다.



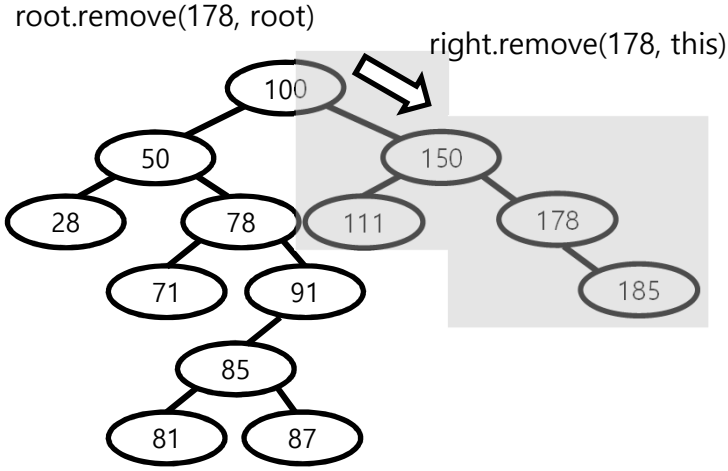
그 다음 94는 78보다 크기 때문에 B문장으로 의해 오른쪽 서브트리에 대해서 재귀호출을 한다.
그 다음 91에 대해서도 비교를 하게 되면 크기 때문에 오른쪽 서브트리에 대해서 재귀호출을 하게 된다.



94에 대해서는 자식이 없기 때문에 D 문장으로 넘어가게 된다. 그래서 child라는 값은 null이 되겠다.(어자피 left도 null인데 right도 null이기 때문이다.) 그럼 현재 parent가 91인 노드라서 이 노드의 우측 자식 노드가 현재 가리킨 노드 94이기 때문에 위의 문장에 의해 94는 결국 null로 없어지게 된다.

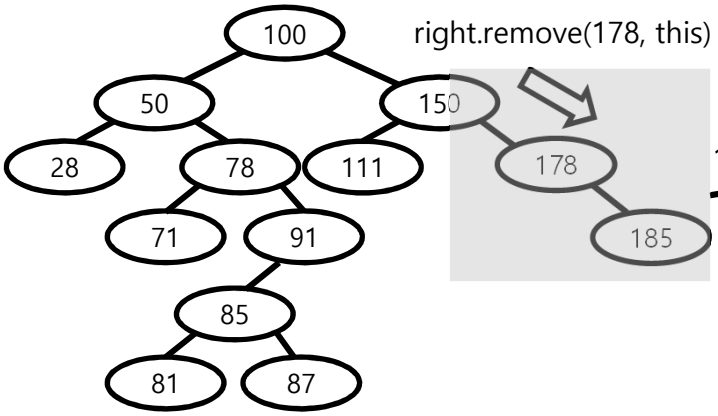
2> 1번 메소드 이후에 178를 삭제한다면...

root.remove(178, root)

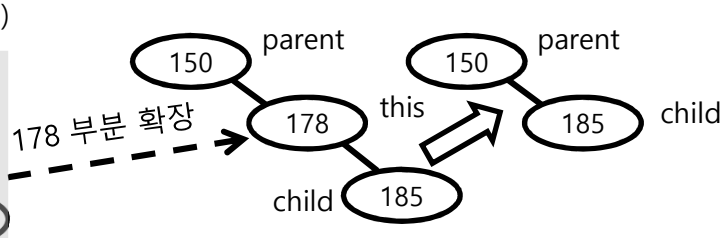


178에 대해서 살펴보면 일단 100보다 크기 때문에 오른쪽 서브트리에 대해서 재귀호출(B문장을 실행) 하게 된다.

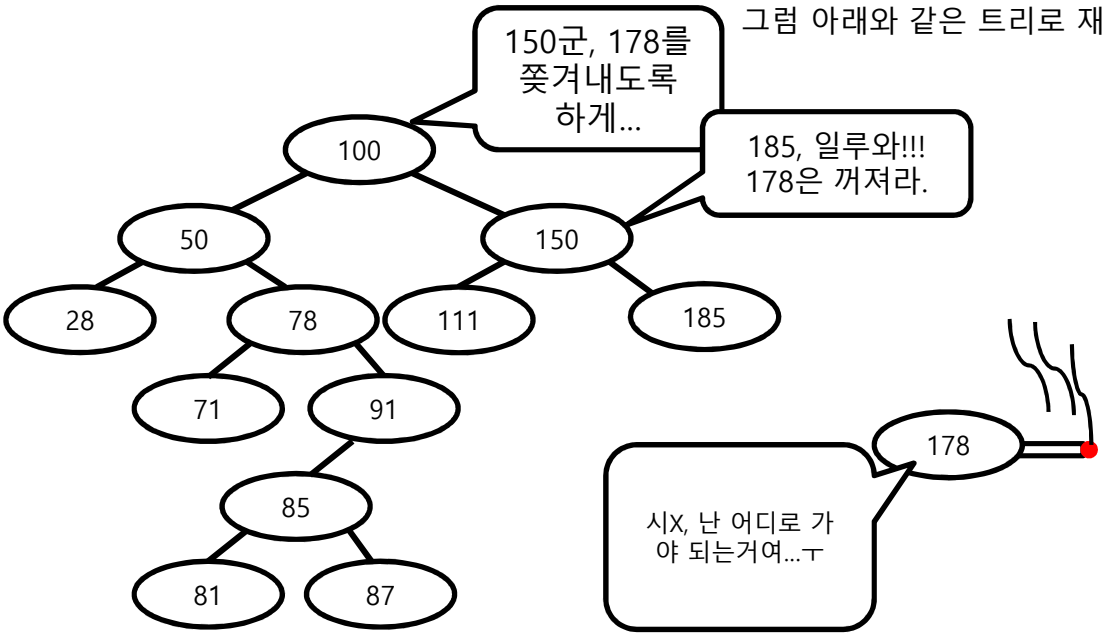
```
public void remove(int value, Node parent){
  if(value<this.value){ // A
    if(left!=null) left.remove(value, this);
  } else if(value>this.value){ // B
    if(right!=null) right.remove(value, this);
  } else{
  if(left!=null && right!=null){ // C
     int temp=right.getLeftMostValue();
     this.value=temp;
     right.remove(temp, this);
  } else { // D
     Node child = (left!=null) ? left:right;
       if(parent.left==this) parent.left=child;
       else parent.right=child;
    }
  }
}
```



178에 대해서 살펴보면 일단 100보다 크기 때문에 오른쪽 서브트리에 대해서 재귀호출(B문장을 실행) 하게 된다. 그러면 어느 새 178이 보이게 될 것이다.

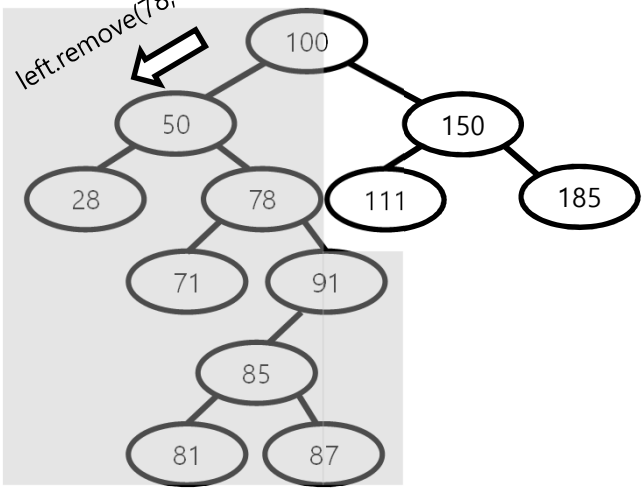


일단 178이란 노드는 자식 서브 트리가 하나(우측 노드에 185) 밖에 존재하지 않기 때문에 D문장으로 넘어가게 될 것이다. 그러면 child는 자동적으로 185란 노드가 되겠다. 그럼 parent가 150이니 자동적으로 else 문 부분을 참조하게 되겠다. parent의 우측 자식 노드가 178이기 때문에 parent의 우측 자식 노드를 185란 노드로 땡기게 되면서 178이란 존재감은 없어지게 된다. 그럼 아래와 같은 트리로 재구성이 되겠다.



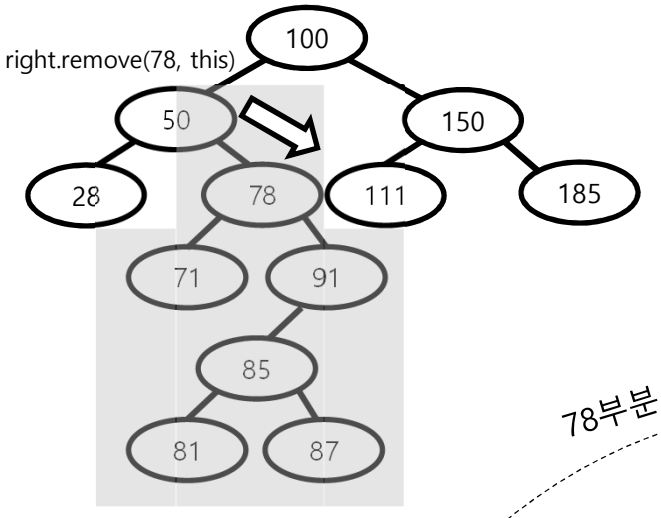
3> 2번 메소드 이후에 78를 삭제한다면...

root.remove(78, root)

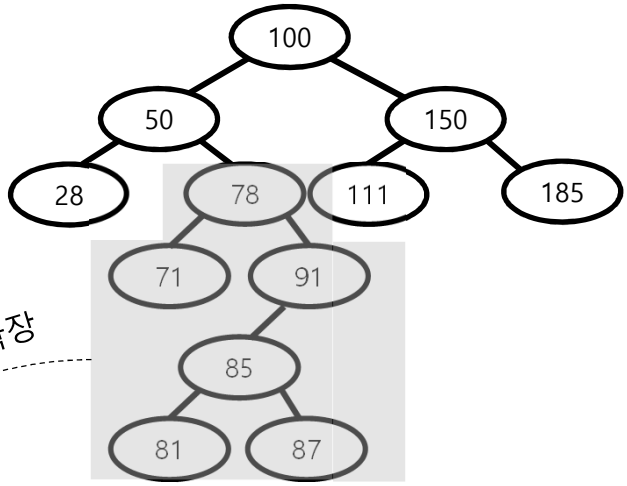


78에 대해서 삭제 작업을 들어가게 된다면 A 문장에 대해서 100의 왼쪽 서브트리에 대해 재귀 호출을 하게 된다.

```
public void remove(int value, Node parent){
  if(value<this.value){ // A
    if(left!=null) left.remove(value, this);
  } else if(value>this.value){ // B
    if(right!=null) right.remove(value, this);
  } else{
  if(left!=null && right!=null){ // C
     int temp=right.getLeftMostValue();
     this.value=temp;
     right.remove(temp, this);
  } else { // D
     Node child = (left!=null) ? left:right;
       if(parent.left==this) parent.left=child;
       else parent.right=child;
    }
  }
}
```



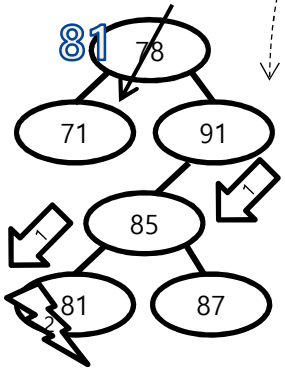
그 다음 78은 50보다 한참 크기 때문에 오른쪽 서브트리에 대해서 재귀 호출을 통해서 B의 문장을 실행하게 된다.



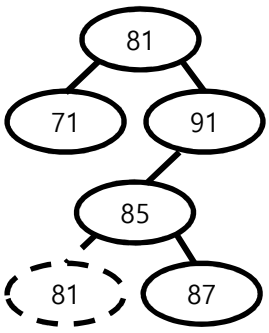
78부분 확장

드디어 78이란 노드를 찾았다. 그렇지만 문제는 78의 자식들은 2명이 있기 때문에 C의 문장으로 넘어가게 된다. 그러면 78의 우측 자식 노드에 있는 값들 중에서 가장 최솟값을 주워와서 78을 없애는 방법을 알아야 된다. 아래와 같은 메소드를 살펴보자.

```
public int getLeftMostValue(){
   if(this.left!=null)  // 1
      return left.getLeftMostValue();
   return this.value; // 2
}
```

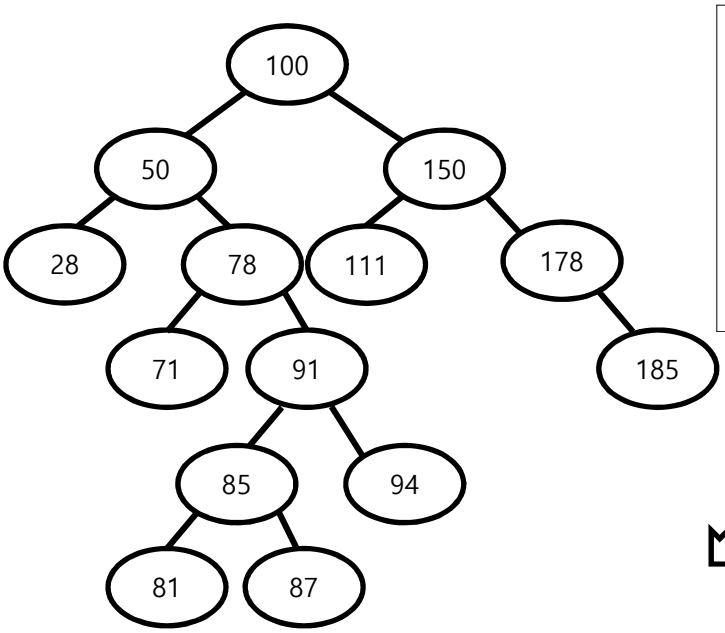


78의 우측 서브트리를 통해서 getLeftMostValue() 메소드를 수행하게 된다면 91부터 시작을 해보겠다. 그러면 91의 좌측 노드가 있어서 재귀 호출을 하게 되어 85를 가리킨다. 그 다음에도 85의 자식이 있기 때문에 81로 내려간다. 그 다음 81은 자식이 없기 때문에 81은 temp의 값이 되면서 78의 자리를 대신해서 81로 채워지게 된다.



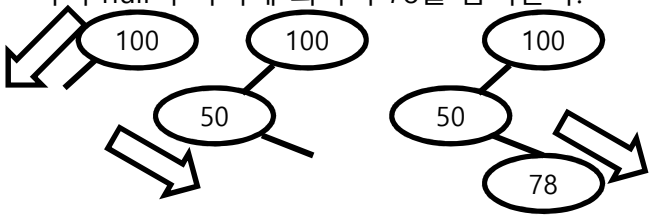
그럼 78이란 값은 이제 없던 일이 되겠다. 그럼 81의 우측 노드를 통해서 옛날의 81의 자리에 대해서 삭제를 하는 작업을 right.remove(temp, this)를 통해서 처리를 하게 된다. 이러한 작업을 통해서 **<u>이진 탐색 트리의 본래 원칙을 지키게 되면서</u>** 최종적으로 삭제 작업이 마치게 된다.

<(첨가2) 이진트리 contains 알고리즘, 어떻게 돌아가는 걸까>




```
public boolean contains(int value){
  if(value<this.value) // A
    return (left!=null) && left.contains(value);
  else if(value>this.value) // B
    return (right!=null) && right.contains(value);
  return true; // C
}
```

예시) 94를 찾는 과정을 살펴보자.
일단 94는 100보다 작기 때문에 A를 먼저 탐색한다. 그 다음은 50에 대해서 비교를 들어가면 오른쪽이 null이 아니게 되어서 78를 탐색한다.



이제 78에도 오른쪽 서브트리가 있으니 다음 값인 91를 탐색을 하고, 91에 나온 값들을 탐색하게 되면 오른쪽 서브트리를 통해서 94를 탐색했다. 그러면 94에 대해서는 A, B에 대해서 아무 것도 속하지 않게 되어서 C라는 문장을 실행해서 이 메소드를 종료하게 된다.

contains는 아시다시피 Collection에 있는 인터페이스의 일부로서 어떠한 값들이 포함되었는지에 대한 알고리즘으로 살펴볼 수 있다. 이 알고리즘을 분석해보면서 어떠한 원리로 돌아가는지에 대해 알아보도록 하자.